THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

SCHOOL OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Generating Evasion Attacks Against Intrusion Detection Systems Using A Genetic Algorithm

By Raymond Mogg

Submitted for the degree of Bachelor of Engineering
(Honours) in the division of Software Engineering

22 June 2020

Raymond Mogg
r.mogg@uq.net.au

June 21, 2020

Prof Amin Abbosh
Acting Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia QLD 4072

Dear Professor Abbosh,

In accordance with the requirements of the Degree of Bachelor of Engineering (Honours) in the School of Information Technology and Electrical Engineering, I submit the following thesis entitled

"Generating Evasion Attacks Against Intrusion Detection Systems Using A Genetic Algorithm"

The thesis was performed under the supervision of Associate Professor Dan Kim. I declare that the work submitted in the thesis is my own, except as acknowledged in the text and footnotes, and that it has not previously been submitted for a degree at the University of Queensland or any other institution.

Yours sincerely Raymond Mogg

Acknowledgements

I would like to acknowledge and thank Associate Professor Dan Kim for his guidance in producing this thesis. His assistance in providing me with the direction of the body of research was invaluable and led me down a path where promising results were not only achievable but valuable. He provided continuous feedback on my methods of attack generation, as well as critical input on the final produced results, which in turn allowed the algorithm to be tweaked and improved.

Abstract

Intrusion detection systems (IDS) are one of the key components that make up modern cybersecurity infrastructure. This thesis investigates the viability of generating interpretable evasion attacks against intrusion detection system through the use of a genetic algorithm. Evasion attacks represent a crafted attack where malicious network packets are classified as benign by an IDS and as such avoids detection, allowing the attack to proceed through and affect a target system.

By first training a decision tree classifier on specific attacks within the NSL-KDD dataset, a decision tree based intrusion detection system was constructed. Then a genetic algorithm was designed that uses the output of the decision tree based IDS as a feedback loop. The fitness function of this decision tree allowed samples to be produced that were similar in structure to a known seed attack but were incorrectly classified as benign by the intrusion detection system, hence producing evasion attack samples. From this, a generalised attack pipeline was then constructed allowing the generation of any attack against any intrusion detection system, as long as standard interfaces are adhered to.

Various recommendations for the design of IDSes were then drawn from the produced results. The recommendations were to ensure no output of the IDS is exposed, limit knowledge of the internals of the IDS to external parties, and finally to have a system in place for picking up repeated inputs of a similar structure by utilising some measure of input deviation.

While this work sets a good base of a potential attack vector against intrusion detection systems, with further work into the validation of these

samples and testing with other potentially more complicated IDSes, it is hoped that current industry-standard IDSes can be improved vastly.

List of Figures

List of Tables

Contents

1 Introduction

Intrusion Detection Systems (IDS) play a critical role in the current cybersecurity ecosystem by monitoring and detecting malicious attacks against computer networks and systems. [1]. While intrusion detection systems help detect and prevent malicious actors from exploiting a system, it is clear that there is still plenty of work needed in the area. The average cost of a breach to a global company is $3.86M USD, and it takes an average of 196 days to identify these breaches [2]. By investigating potential attacks against IDSes, improvements in their architecture can be made to strengthen their intrusion detection capabilities, leading to a decrease in the impact of cybersecurity attacks.

This research proposes techniques and a generalised attack pipeline allowing the generation of interpretable evasion attacks against any black box IDS through the use of genetic algorithms. By leveraging the NSL-KDD dataset [3] and focusing on two attack types, attack samples were successfully produced that were able to evade a decision tree based IDS. Since both the samples and the IDS used to classify samples were interpretable, human analysis was conducted on produced samples to evaluate their effectiveness. From these evaluations, conclusions were able to be drawn that may enable the strengthening of current IDSes.

The genetic algorithm pipeline was also designed in a modular way, such that it can be reused with any IDS by wrapping any classification model in a simple Python interface. This design allows other researchers to utilise the algorithm against potentially more complex IDSes. It is hoped that this knowledge, paired with the attack generation pipeline, can be utilised in further research to continue to strengthen current IDSes.

1.1 Aims

The aim of this project is to generate interpretable evasion attacks against intrusion detection systems using genetic algorithms. The IDS will be network based, and use a behavior based detection method. The final produced work will be a generalised attack pipeline capable of producing attacks against any IDS that conforms to a set interface.

1.2 Scope

1.2.1 In Scope

The following items will be produced as part of this thesis:

- A decision tree based IDS, using behavior based detection as its detection method and utilizing a network based audit source.
- A genetic algorithm pipeline to generate attacks
- Interpretable attack samples for the two attack types mentioned in **Section 1.2.3**
- Analysis of attack performance
- Recommendations on how intrusion detection systems can be improved to avoid these types of attacks

1.2.2 Dataset

The NSL-KDD dataset [4] will be used for producing both the attack classifier (decision tree based IDS) and for generating attack samples as part of the genetic algorithm. This dataset was chosen due to its wide selection of data, recent production, and extensive use in other reserach papers.

The NSL-KDD dataset was produced to solve some issues with the KDD99 dataset, such as redundant records and a low difficulty level of attacks [4].

There are 22 attacks included as part of the NSL-KDD dataset [4], including - neptune, warezclient, ipsweep, portsweep, teardrop, nmap, satan, smurf, pod, back, guess_passwd, ftp_write, multihop, rootkit, buffer_overflow, imap, warezmaster, phf, land, loadmodule,spy, perl

The attacks from the NSL-KDD dataset can be broken down into the following attack classes, summarized in **Table 1** [4].

Table 1: Attack Class Breakdown

Attack Class	Attacks
DoS	Smurf, Land, Pod, Teardrop, Neptun, Back
R2L	Ftp write, Guess pass, Imap, Multi-hop, phf, spy, warezmaster, warezclient
U2R	Perl, buffer overflow, rootkit, loadmodule
Probe	Ipsweet, nmap, portweep, satan

1.2.3 Attack Types

In order for attacks to be effectively produced and keep the scope sizeable, only two attack types will be investigated, however, the attack pipeline will be usable on any attack present in the NSL-KDD dataset. The two attack types that are being focused on are the teardrop attack and nmap probe attack. These two attacks represent two different attack scenarios - with one being a denial of service and the other being a probe, which are two commonly occurring attacks seen today. Targeting different attacks will show that the pipeline can be used to generate attacks of any type given enough training data is present. [5].

1.3 Relevance

This research is highly relevant in the current cybersecurity ecosystem, with the number of unique cybersecurity attacks up 27% in 2018 [6]. By exploring how attacks can be generated that could potentially evade an IDS, measures can be put into place to improve the design of current IDS, essentially patching for these types of generated attacks. While this patching will be a difficult task, an understanding of what such generated attacks may look like will still aid in the further research of how these types of crafted attacks may be prevented. The fact that the attack generation pipeline has been modeled in a modular way means that any researcher can plug in their own IDS model and generate attacks against it. This will allow further research into the weaknesses of current IDSes.

1.3.1 IDS

In order to preserve interpretability, a decision tree based IDS has been utilised. While there are many ways to ensure machine learning models are interpretable, by utilising a model that is inherently interpretable, the IDS can be easily produced and analysed without the need for other complex systems to aid in the production of its interpretability. [7].

1.4 Results

The proposed attack generation pipeline was successfully able to generate a decision tree classifier based on the NSL-KDD dataset, and then use this classifier as part of the feedback loop in the genetic algorithm. Potential evasion attack samples were able to be produced by the genetic algorithm for both the teardrop and nmap attacks being investigated. These attack samples were classified as benign by the IDS, but were very similar in structure to the attack sample used to seed the algorithm. For the teardrop attack, results were

produced that were able to find the decision boundary of the IDS successfully, and as such shows that there is potential for this attack generation method to be extended to other more complex intrusion detection systems with success.

In future work, these produced samples may be validated to produce confirmation that they are actual attack samples that were misclassified as benign. With this work, full validation of the produced results here would be possible.

2 Background

2.1 Intrusion Detection Systems (IDS)

IDSes exist in order to monitor various computer systems and detect potential attacks against these systems [1]. Attacks can come both in the form of misuse by legitimate users and in the form of malicious attacks by third parties [1]. An IDS can be broken down into 3 main components that define its area of operation. These are its detection method, behavior upon detection, and audit source location [1].

2.1.1 Detection Methods

There are two main detection methods predominantly used within current IDSes - behavior based and knowledge based detection [1]. Behavior based detection systems use pattern matching and contain a database of known attack vectors. By looking at the fingerprint of a potential attack, it can be checked against the known attack database to clarify whether this is an attack or not. This is referred to as misuse detection [1].

Behavior based detection systems work by creating a model of normal user behavior. Any behavior that falls outside this model of normal behavior is then classified as an attack. This is referred to as anomaly detection [1].

Due to the nature of the two systems, knowledge based IDS's are extremely effective at detecting known attacks, as they often contain a vast amount of information about potential attacks [1]. The downfall to this is that any action by a user that does not map to a known attack is considered acceptable behavior. This means that any newly crafted attack or a well-known attack that deviates slightly from the attack pattern may be classified

as normal behavior and no alarms will therefor be raised [1]. As such it is imperative that knowledge based IDS's have an up to date database of attacks.

Behaviour based IDS's on the other hand are able to detect new and unseen vulnerabilities as they simply base their detection method of the deviation from normal behavior [1]. This however has the drawback that they may produce a high false-alarm rate, due to unseen behavior that is not necessarily an attack but falls outside the normal behavior range being classified as an attack. The following table highlights both the pros and cons of each detection method.

Table 2: Pros and Cons of Detection Methods

Detection Method	Pros	Cons
Knowledge Based	• Low false alarm rate • Good detection of known attacks	• Poor detection of unknown attacks • Difficult to gather a complete database of known attacks • Maintenance of attack database can be difficult and time consuming
Behaviour Based	• Good detection of unknown attacks • Can detect "abuse of privilege" type attacks	• Higher false alarm rate • Behaviour changes over time, leading to the need to retrain the model of the IDS

2.1.2 Audit Source

The audit source of an IDS defines where the data it processes comes from. There are two main audit sources used in current IDS's - Host based and Network based. In recent times there has also been an increase in hybrid systems that use a combination of host based and network based audit sources in order to improve detection capabilities and performance [8].

Host based audit sources use information such as logs from host systems in order to monitor and detect potential attacks, while network based audit sources use network packets and traffic.

While host based audit sources were the first to be used [1], with an increase in network based attacks such as DoS attacks, network based audit sources are now needed to be able to detect a multitude of attacks [1].

2.1.3 Behavior On Detection

Behavior on detection within an intrusion detection system is defined as the action taken once a potential attack is detected. There are two main types of possible behavior - active and inactive. Active IDSes respond to potential attacks proactively by either logging potential attackers out of the system, or by taking corrective steps to fix the issue [1].

Inactive IDS's simply trigger alarms which are then acted upon by humans [1]. These passive systems have the trade-off that a large amount of damage may already be done before any action can be taken

2.2 Interpretability in Machine Learning

Interpretability is the concept of producing machine learning models in which it is possible to meaningfully inspect specific components of the model and understand how inputs were processed by the model [7]. This provides great value to not only designers of systems (in this instance, designers of intrusion detection systems) but also users of the system, as they are able to understand why their actions or input lead to a specific outcome.

There are three main advantages drawn by using interpretable models in a system [7]:

- Understanding how the model draws conclusions produces more trust in the system over a black box solution.
- The model is more easily able to be improved, as instances where it performs poorly can be closely analysed and understood by a system designer.
- The model is much more useful to a user when its decisions are interpretable. Take the example of recommending music - a recommendation can be made with context, for example, "You may like artist A as you like artist B" over simply having the recommendation "You may like artist A"

There are currently multiple approaches to tackling the interpretability problem in machine learning. The first approach is to use models that are inherently interpretable themselves, such as decision trees, sparse linear models, and rules [7]. The other approach is to use model-agnostic interpretability in which model information is extracted by either training another model on the outputs of the original model itself [9], or by perturbing inputs and seeing how the model reacts [10].

Since the main user of the produced attack generation pipeline is researchers, the ability to have an interpretable model will allow for the produced work to be much more useful in a research setting as the decisions of the IDS will be very transparent.

2.3 Attacks Against IDS

Since IDSes are themselves a computer system, they are vulnerable to attacks and exploitation [11]. There are a few classes of attacks commonly used against IDSes.

2.3.1 Evasion Attacks

Evasion attacks are a type of attack in which an adversary carefully crafts their attack to ensure that the pattern picked up by the IDS is not classified as an attack, even though it still is malicious [11]. There are two classes of evasion attacks depending on what detection method the IDS is using. If the IDS is using a knowledge based detection system, the evasion attack aims to modify the attack enough so that the system does not register its footprint with one stored in the knowledge base. If the IDS is using a behaviour based detection system, then the evasion attack aims to mimic normal behavior despite being malicious (mimicry attack) [11].

2.3.2 Poisoning

Poisoning attacks stem from the fact that many modern IDSes use some sort of machine learning to train their models [11].

In this attack, well-crafted patterns are added into the dataset used for

training models such that the algorithm produced will be biased in some way [11].

2.3.3 Denial of Service

A Denial-of-Service (DoS) attack is when network traffic is used to overwhelm a system. In this attack an attacker may employ various server weaknesses or simply use pure throughput to take down services, rendering the IDS useless as the system is completely down [11].

2.3.4 Overstimulation

Overstimulation attacks are similar to DoS attacks in that they attempt to overwhelm the resources of a system. In an overstimulation attack, an attacker generates a large number of attack patterns in an attempt to generate many attack alerts within the IDS, in turn, overwhelming security personnel, other analysers, or the entire alerting system itself [11].

2.4 Genetic Algorithms

Genetic algorithms utilise the biological idea of evolution to produce optimal solutions to a set problem space. They produce offspring that are bred to perform against a well-defined fitness function. [12]. There are five key components within a genetic algorithm that need to be identified [12], which are:

- A genetic representation of the solution space; Each potential solution to the problem space must be able to be represented in a genetic way.
- An initial population; To begin the process an initial population is needed. This population can be sampled from some known subset of

data, or generated from a seed sample. The fittest will then be selected and create the next generation.

- A fitness function; A fitness function is needed in order to evaluate the quality of a given solution. This is how the best individuals are chosen to reproduce.

- A function for producing offspring; The function must take in two individuals and produce either a single or a set of individuals. Through this process, genetic mutation and crossover is introduced which allows the population to evolve from generation to generation.

- A selection function; A function that selects the fittest offspring from the population, who then proceed to the next round.

By producing a genetic algorithm with the above five components, attack samples can be generated and then evaluated against an IDS in order to gauge their fitness. From this, the best attack samples can then be used to generate further attack samples.

2.5 Related Work

The following research articles are closely related to the area of this research. An analysis has been performed on these articles to identify their scope, similarities and differences between the research, and how they can be used to guide this thesis.

2.5.1 IDSGAN: Generative Adversarial Networks for Attack Generation against Intrusion Detection

This paper introduces a new method for generating attacks against intrusion detection systems by utilising generative adversarial networks (GANs) [13].

The goal of this model is to be able to produce malicious traffic samples that can evade detection from a black box IDS [13]. The dataset used by the proposed model is the same dataset proposed in this body of work and as such many similarities can be drawn between the two branches of research.

Their proposed IDSGAN model works as follows

- Malicious traffic is pulled from the NSL-KDD dataset and mixed with noise. This is then fed into the generator section of the GAN which generates the adversarial traffic.
- Adversarial traffic and normal traffic is fed into the black box IDS.
- This traffic with predicted labels is then fed into the discriminator of the GAN, which is responsible for calculating the loss of the generator after classifications have been provided by the IDS.
- This loss is then fed back into the generator and used to improve attack generation

By modeling both the generator as well as the discriminator as neural networks, a feedback loop is able to be placed between them. The discriminator learns from the output of the IDS, essentially modeling the black box IDS, while the generator uses this information to improve its generation of malicious network samples by using the discriminators feedback to train the weighting of its neural network [13].

This allowed the team to be able to generate attacks against intrusion detection systems quite successfully. One such metric produced was that using their model, only 10.49% of adversarial attacks were able to be detected by a decision tree based IDS.

2.5.2 Generating attacks and labeling attack datasets for industrial control intrusion detection systems

This PHD thesis covers three components of a framework that can be used to aid in the understanding of attacks against IDSes deployed for critical infrastructure, as well as being used to generate attack datasets to train future critical infrastructure IDSes. Of significant importance to this thesis is the attack generation framework used, as a similar technique can be applied to generate IDS attacks using genetic algorithms.

To produce a SCADA attack generation framework, this paper utilizes a modular approach and first breaks down attack generation into ten requirements [14]. Of importance to this thesis are the following components:

- Ability to parse SCADA protocol messages
- Ability to replicate the SCADA protocol stake
- Ability to sniff local SCADA network traffic
- Ability to inject anomalous SCADA protocol messages into the network.
- Ability to modify protocol message data in real-time
- Ability to flood a SCADA service with anomalous messages

While all of these items are specific to attack generation for SCADA systems, the above functionality can be generalised and then modeled towards the two attack types being investigated in this thesis and form a good basis for the development of attack generation.

2.5.3 Mimicry Attacks on Host-Based Intrusion Detection Systems

This article covers generating mimicry attacks to be used against host based IDSes, with many similarities able to be drawn to our network based approach. The article first outlines six key mimicry attack types, which are summarized in Table 3 [15]. While many of the summaries relate to host based IDS and not behavior based, they may still be applied to this thesis. For example, the insert no-ops attack may be used in a network based system by simply adding in extra network traffic, which may hide the attack as normal behavior and hence avoid detection by the IDS. The generate equivalent attacks method is the method that will be drawn on most for the produced algorithm in this work. By getting the genetic algorithm to produce samples similar to a known attack but with slight variations, it is hoped that a malicious attack still exists but is classified as benign by the IDS.

Table 3: Mimcry Attack Type Summary

Attack Type	Summary
Slip under the radar	Avoid causing any chaos in the infiltrated application, since some IDSes only detect attacks via their call signature within the program.
Be patient	Wait for a time when the attack can be executed without raising any alarms (assuming one exists)
Be patient, but make your own luck	Wait for a time when the attack can be executed without raising any alarms, but instead of simply passively waiting, nudge the application into running on the desired path for the execution of the attack
Replace system call parameters	Many IDS do not look at system call parameters. By replacing parameters, a benign system call can be made a malicious one.
Insert no-ops	Padding the attack with no-ops (operations that do nothing) may class the attack as normal behavior and hence avoid detection.
Generate equivalent attacks	There are many ways to craft a malicious attack. By varying the attack sequence even slightly, detection can be avoided.

The article then goes on to detail a formal approach for generating attack samples by investigating the intersection of all sets of attacks with the intersection of the set of all sequences of system calls that do not trigger an alarm within the IDS. While this formal approach translates nicely to behavior based IDSes that may only look at the last six system calls [15], this approach does not scale to network based systems where an attack vector may consist of hundreds or thousands of network packets. As such this is where genetic algorithms will instead be used as part of this thesis, to generate samples that ultimately can avoid detection.

3 Proposed Approach

3.1 Methodology

In order to achieve the aims of this research, the following methodology was utilised. This methodology not only allowed results to be generated in the form of attack samples, but it also allowed a generalised attack pipeline to be built out in a modular way.

- Construct a decision tree based IDS that can be trained on the NSL-KDD dataset.
- Train and test the decision tree IDS.
- Construct the genetic algorithm that uses an NSL-KDD attack sample as a seed.
- Develop an interface that allows the IDS produced and any other IDS to be integrated into the genetic algorithm, allowing the feedback loop between the IDS and the genetic algorithm to be used to generate samples.
- Construct a testing library that allows the testing of various portions of the genetic algorithm and the decision tree.
- Link components together to produce an overall attack pipeline that given a single entry point and some variables, can generate the decision tree and attack samples all at once in an efficient manor.
- Conduct hyperparameter tuning to allow the algorithm to be optimised to produce attack samples to evade a Teardrop and NMAP IDS.
- Produce results using previously mentioned libraries and analyse the performance of the algorithm

This methodology was able to be successfully utilised and as described in the following sections, all components were built out and tested.

Since the final produced work is a piece of software that allows attack generation, proper software engineering principles were followed in the development of the system. Modular code was produced to allow other IDSes to be used. Python was chosen as the primary software language due to its ease of use and a vast selection of open source data science tools available.

The libraries used as part of this work are as follows:

- Matplotlib: Used for plotting and visualisation of produced results and experiments that were conducted.
- Numpy: Used for data generation in the testing of the attack generation pipeline.
- Pandas: Used for pre-processing data and data manipulation.
- Sklearn: Used to generate decision tree classifiers to be used as the IDS in the attack generation pipeline.

3.2 Overall Attack Pipeline

The following outlines the overall attack pipeline produced as part of this research. The attack pipeline contains all individual components required to generate sample evasion attacks and can be run using samples presented in the interface section. See appendix A for the codebase and more on running the attack generation pipeline locally.

The steps taken by the attack pipeline are as follows:

- Train a decision tree based IDS on the provided attack type
- From the NSL KDD dataset, select a seed attack to be used by the algorithm. The seed attack is a random sample from the dataset that is the attack type being investigated.
- Start the genetic algorithm using the produced IDS and seed attack.
- Run the genetic algorithm for 20 generations. Each generation contains 120 samples, with the fittest 30 of those samples (As evaluated by the algorithms fitness function) moving to the next generation.
- Final 30 fittest samples are collected and classified

At the conclusion of this process, the fittest sample is considered the best candidate for an evasion attack sample as it is most similar to the seed attack, however it is still classified as benign. The following figure outlines the modules involved in the above process, including the feedback loop between the IDS and the genetic algorithm.

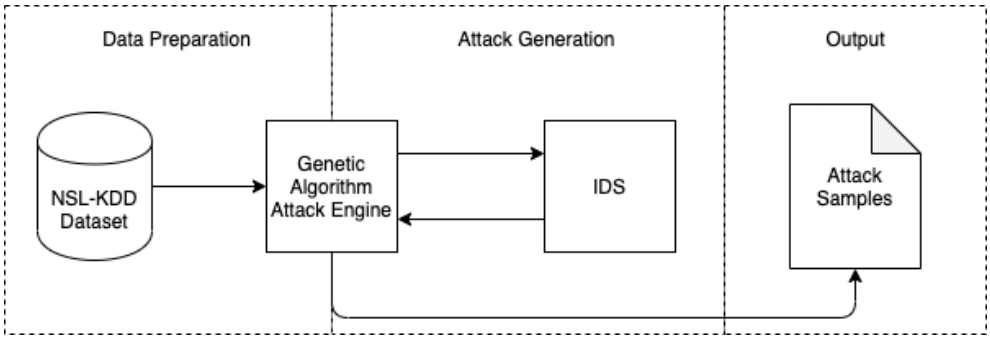

Figure 1: Overview of the attack pipeline

3.3 Decision Tree Based IDS

A core part of the produced work is the decision tree based IDS. This IDS is used for the classification of samples, as well as in the overall attack generation

pipeline as a feedback loop to gauge the fitness of individual samples. While real-world IDSes are commonly more complex than decision trees, the choice of using a decision tree as the IDS for this research centered around the fact that they are highly interpretable and as such results can easily be analysed.

3.3.1 Testing and Training

In order to train and test the model, a regular train test split was used. with a train test split of 17%.

The methodology that was used for conducting the training and testing of the decision tree is as follows

- Train and test datasets are loaded into Pandas Dataframes
- An attack label is created, which is a binary column representing if the sample is the attack the model is being trained on, or not. If the sample is an attack other than the attack being investigated, this column's value is 0.
- The protocol type, service, and flag columns are dropped from both the training and testing sets
- The attack and attack label columns are dropped from the training dataset. The attack label column is used as the target for the training.
- Decision tree is trained. If the test model environment variable is set, the model is then evaluated. If not, the model is returned (for use by the genetic algorithm

3.3.2 Interpretability

To preserve interpretability, when the decision trees were trained, their max height was set to 5. This limits the number of levels produced by the tree.

While this may decrease overall classification performance, it allows a human to analyse the decision boundaries of the tree, as desired for this research.

3.4 Genetic Algorithm

The genetic algorithm is the second key component used in the attack generation pipeline. It is responsible for producing the final attack samples.

There are a few key constants used by the algorithm that were fine-tuned for optimal results. See the next section for hyperparameter tuning for more on this. The following are the final values used for the algorithm constants to produce results.

- Mutation Percentage - the percent chance that an individual feature variable mutates in any given offspring: **18%**
- Iterations / Generations - The number of iterations the algorithm is run for. Can also be considered the total number of produced generations: **20**
- Offspring - The total number of offspring to produce at each iteration: **120**
- Fittest Offspring - the total number of offspring to live and breed the next generation. Selected in descending fitness order: **30**

3.4.1 Genetic Representation of the solution space

In order to represent the solution genetically, each feature variable within the NSL-KDD dataset is being considered a single gene. This means each feature variable can be mutated on an individual basis throughout the algorithm, influencing the final produced sample. Some constraints were placed on which

feature variables could be mutated to preserve feature variables critical to the attack type. A limit was also placed on the range of feature variables to again ensure that the produced samples were viable.

3.4.2 Genetic Mutation

Genetic mutation is used to simulate genetic drift amongst the population, allowing the population's overall fitness to increase over time. The genetic mutation percentage parameter is used to influence how much genetic mutation should occur at each iteration of the algorithm for each gene. This is implemented by looking at each feature variable on an individual basis. A random number between 0 and 100 is generated for each feature variable, and if this value is less than or equal to the genetic mutation percentage, that feature variable is mutated for that sample. If a mutation occurs, the value of that feature variable is set to be a random value between the maximum and minimum supported values for that feature variable, chosen from a uniform random distribution between the two limit values. Other methods could be applied here such as adding a small deviation to the current feature variable value, which would lead to slower but potentially more fine-tuned mutation.

In order to ensure that a sample is not changed in excess, causing it to no longer be a valid attack of the type being investigated, restrictions were added to limit which parameters can be mutated during the mutation phase. This ensures that if the algorithm does not change fields that should not realistically be changed for this attack type. The fields constrained were:

- Protocol Type
- Service
- Flag

While these fields ensure that the attack remains consistent with the attack type being generated, further work in this area could ensure that generated attacks are valid attack samples. To do this, the exact fields that need to remain consistent need to be identified for each attack type. One such method of doing this as described in the production of the IDSGAN [13] is to characterise the properties of the feature variables based on the attack types and only mutate feature variables that are not integral to that attack type.

3.4.3 Fitness Function

The fitness function is used to evaluate the quality of individual samples. In order to devise the fitness function, the goals of the algorithm must first be outlined. The goal of this algorithm was to produce samples that were classified as benign by the decision tree based intrusion detection system, but the characteristics of their data meant that they are still attacks. Since there was no mechanism as part of the attack pipeline to validate whether a sample is an attack, this goal can be achieved by generating benign samples that are as similar to a known attack as possible, while still ensuring that they are classified as benign by the IDS. This problem can be rephrased as finding the decision boundary of the decision tree classifier. From this, the fitness function was produced as the following, where f represents the fitness of a sample and d represents the deviation of a sample.

If the sample is classified as benign:

$$f = 2 * (1/d) \tag{1}$$

If the sample is classified as an attack:

$$f = (1/d) \tag{2}$$

Where deviation is given by measuring the sum of the difference in each feature variable from the given sample to the attack sample that seeded the algorithm. Given s represents the attack sample that seeded the attack, g is the given sample that deviation is being calculated for, d is the deviation of the samples, and the subscripts of the equation indicate the index of the feature variable:

$$d = \sum_{n=1}^{41} abs(s_n - g_n) \tag{3}$$

The scalar value used as part of the benign sample fitness function is present to ensure that benign samples are more favoured by the algorithm than attack samples. The inverse of deviation was then used as this ensures that deviation is minimised.

As part of future work, it is also proposed that the values used to scale the output of the deviation in both fitness function equations are moved to two separate hyperparameters, Fa and Fb respectively. These parameters can then be fine-tuned to achieve more optimal results, rather than using an arbitrary value of 2 and 1. By moving these values closer together while still rewarding benign samples, it is theorised that attacks with less deviation may be able to be produced, hence better finding the decision boundary of the IDS.

3.4.4 Initial Population

The initial population of the algorithm was produced by taking the seed attack and breeding it with itself to produce an initial population of the required population size of 120. Genetic mutation is introduced as part of this initial breeding, which means that the initial population already has some genetic drift injected into it. From this initial population, the rest of the population is bred. This also means that within the initial population

you will have a mix of samples classified as both benign and attacks. Another potential approach to the used method would be to simply pick 30 attack samples from the NSL-KDD dataset to use as the initial population, and pick one at random as the seed sample.

3.4.5 Breeding Function

In order to breed each generation, the breeding function is used. This function takes in two samples, chosen randomly from the current population, and produces a new sample by conducting the following steps

- With equal probability, produce a new sample by picking each feature variable from either parent 1, or parent 2 and using it as part of the new sample.
- Add genetic mutation on a gene by gene bases by sampling a number between 0 and 100 for each gene (feature variable) of the sample. If this number is less than the genetic mutation percentage (18), then mutate the gene.
- Mutate each gene as required by picking a new value for that gene that is within the maximum and minimum value for that feature variable. This new value is chosen randomly with equal distribution across the maximum and minimum value.

3.4.6 Optimisation Techniques

A few optimisations techniques have been presented throughout the individual components of the produced genetic algorithm, and are summarised as follows:

- Parametrise the scaling values within the fitness function. This will allow further hyperparameter tuning to be conducted on these variables, further optimising the fitness function.

- Introduce more feature variable locking for different attack types. By further limiting what feature variables may be changed for a given attack type, more realistic attacks may be produced as their form will be more similar to a known attack while still being mutated enough to be classified as benign.

3.5 Code Structure and Interfaces

As seen in figure 1, the design of the attack generation pipeline follows a modular approach, where most components in the system can essentially be modified as long as it conforms to the standards of the other components. While the genetic algorithm and NSL-KDD dataset are tightly coupled and as such quite difficult to swap out without re-writing lots of code, the IDS is highly modular and can be easily exchanged for a completely different IDS. The attack generation pipeline was designed in this way to allow for potential further research to be conducted on generating attacks against intrusion detection systems using the same genetic algorithms, but with different IDS systems in place which may produce differing results. It is hoped that this may allow the analyse of how more complex IDSes may perform against evasion attacks generated by genetic algorithms.

In order to conform to the interface that is consumed by the genetic algorithm, a wrapper class can be written for any classification model, as long as the wrapper class implements the following basic interface:

This means that any classification model from any library can be used as part of the model, opening up the option for the exploration of much more complex models to be used as an IDS and investigating whether attacks are still possible against these models.

```
1       class ModelWrapper:
2
3           """
4           Predict the class of a value(s) X
5           Returns an array representing the predictions
6           for each value in X, where
7           1 = sample classified as an attack and
8           0 = sample classified as benign
9           """
10          def predict(x):
11              #Run model prediction
12              return pass
```

Figure 2: IDS Wrapper Interface

3.5.1 Code Structure

The following is a description of the produced modules that make up the attack generation pipeline.

- GeneticAlgorithm.py: Defines all the genetic algorithm code for running the algorithm. It requires a model to be able to run.
- Model.py: Defines all the code to produce the decision tree based IDS that is used as part of the genetic algorithm. Produces a model that conforms to the standard ModelWrapper interface.
- AlgorithmTesting.py: Provides helper classes to run various tests using the genetic algorithm pipeline. These tests were used to produce results as part of this body of work.
- labels.py: Contains all header and data structure info for the NSL-KDD dataset.

```
1       def runAlgorithm():
2       #Model that adheres to the ModelWrapper interface
3       model = Model()
4
5       #Debug=false, mutationPercentage=18
6       #attack=teardrop, saveModel=true, model=model
7       algorithm = GeneticAlgorithm(False, 18, "teardrop", True,
            model)
8
9       #Run the algorithm for 20 iterations
10      #120 offspring per generation
11      #0 survive each generation
12      finalPopulation = algorithm.run_algorithm(20, 120, 30)
13      return finalPopulation
```

Figure 3: Sample Algorithm Code Usage

3.5.2 Using the algorithm

In order to utilise the algorithm as part of your code, you simply need to import the genetic algorithm class and intiilise it by supplying a mutation percentage, name of the attack to target and your own model (if you wish to use a model other than the decision tree classifier provided). You also need to pass in flags for if you want to run in debug mode, and if you want to save the model. Once the model is initilised the algorithm can be run by using the runAlgortithm function, along with parameters defining the number of iterations to run for, the number of offspring to produce per iteration, and the number of offspring to survive each round. The following is a sample code snippet of a basic algorithm run.

For more on using the algorithm, see the github repository or use the AlgorithmTesting file.

4 Results

4.1 Hyper Parameter Tuning

To produce quality results on the NSL-KDD dataset, hyper-parameter tuning was conducted to find the optimal operating conditions for the algorithm. The produced hyper-parameter values may only provide optimal performance to the algorithm when used in conjunction with the NSL-KDD dataset and on the two attack types focused on in this research. The values used do however serve as a good base point for producing results for other attacks and other datasets.

All the following tests were conducted using the teardrop attack as a point of reference. All references to an attack sample are samples produced that are classified by the IDS as an attack.

4.1.1 Genetic Mutation Parameter

The genetic mutation parameter controls the amount of mutation applied to each feature variable in a given sample. In order to test the effect of the genetic mutation variable on the produced attacks, the following test was conducted.
Vary the genetic mutation variable from values 0% to 50%. For each value used, record the maximum and minimum fitness function values, as well as the number of attack samples and benign samples produced at the end of running the algorithm. The following details the results.

Mutation Percentage	Max fitness value	Min fitness value	Samples classified as an attack	Samples classified as benign
0	79.985	79.985	10	0
5	38.366	37.579	10	0
10	44.574	3.160	9	1
15	3.869	0.947	7	3
20	1.013	0.528	4	6
25	0.608	0.406	1	9
30	0.454	0.330	0	10
35	0.427	0.285	2	8
40	0.348	0.240.	0	10
45	0.373	0.217	1	9
50	0.301	0.195	0	10

Table 4: Genetic Mutation Testing Results

As can be seen from the above data, the genetic mutation percent variable has a linear effect on the number of benign samples produced as well as the minimum and maximum value for fitness. The fitness value can be seen to converge as for any given seed attack, there is a maximum fitness value (given by minimum deviation and the sample being classified as benign) which depends on the attack used to seed the algorithm. For higher mutation percentages, while more benign samples are produced, these samples can be considered a form of over-fitting. This is due to the fact that a high mutation percentage means each sample mutates very frequently, leading it to differ from the original attack sample vastly. Due to this, an optimal value of 18% percent was chosen, as it sits between the 15% and 20% values which is where the optimal performance of the algorithm appears to occur.

4.1.2 Iterations Parameter

The iterations parameter defines how many iterations (or generations) the genetic algorithm runs for. By increasing this value from 10 to 100 in steps of 10, its effect on the output can be simulated. To do this, the genetic mutation percent, samples per iteration, and offspring number. were fixed at 20%, 20 and 10 respectively.

Number of Iterations	Max fitness value	Min fitness value	Samples classified as an attack	Samples classified as benign
10	0.684	0.472	9	1
20	2.559	0.787	2	8
30	1.019	0.609	3	1
40	1.291	0.666	4	6
50	1.842	1.007	3	6
60	1.149	0.696	0	10
70	1.500	0.842	6	4
80	2.725	1.138	4	6
90	1.442	0.916.	0	10
100	10.022	0.907	3	7

Table 5: Iterations Testing Results

As can be seen from the above results, the number of iterations the algorithm runs for does not directly influence the fitness function or the number of benign samples produced. As such 20 has been chosen in order to allow the algorithm to run in a more time-efficient manner.

4.1.3 Number of Offspring and Fittest Offspring Parameters

The number of offspring and fittest offspring parameters define how many offspring to breed each generation and how many of those bred move onto

the next generation to breed respectively. To test these two parameters and their interaction between each other, two tests were conducted.
First, the ratio between the two was fixed, and the values were scaled linearly. To do this, the ratio was set as the following, where n is the total number of offspring produced each round, and f is the fittest offspring who move through to the next round

$$n = 2 * f \tag{4}$$

In essence, each generation, half the offspring are killed off and half move through to breed.

Number of Fittest Offspring	Max fitness value	Min fitness value	Samples classified as an attack	Samples classified as benign
10	1.538	0.812	3	7
20	10.261	0.600	9	11
30	3.0	0.548	12	18
40	3.992	0.581	12	28
50	13.180	0.636	25	25
60	43.0	0.581	14	46
70	1.321	0.559	22	48
80	9.728	0.587	23	57
90	7.197	0.618	29	61
100	5.375	0.573	36	64
110	6.725	0.592	36	74
120	3.741	0.587	49	71
130	2.891	0.545	42	88
140	13.876	0.595	44	96
150	50.0	0.584	49	101
160	14.285	0.597	44	116
170	5.463	0.582	74	96
180	3.207	0.626	59	121
190	14.285	0.616	63	127

Table 6: Linear Offspring Testing Results

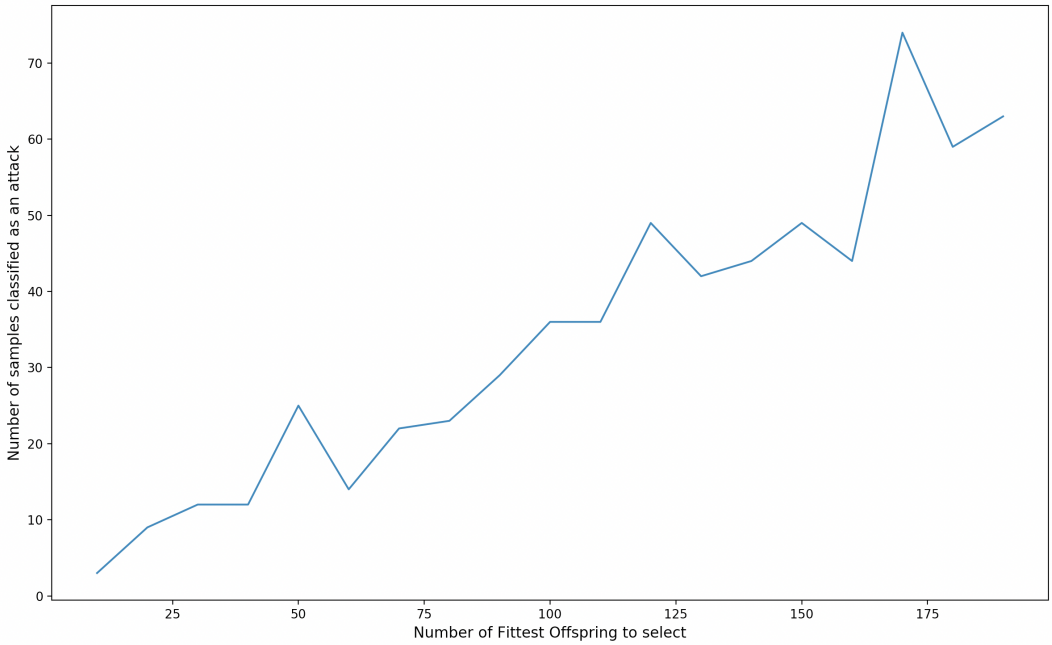

Figure 4: Number of fittest offspring vs samples classified as an attack

As can be seen from the above data and figure, the number of samples classified as an attack scales linearly with the number of offspring produced and the number of fittest offspring to select. This also means that the number of benign samples produced scales linearly with this increase as the number of benign samples is simply the total number of samples (which increases linearly) produced minus the number of attack samples produced.

The next test aims to see the effect of the ratio between the two values. To do this, the number of fittest offspring was fixed to 30, and the total number of offspring per generation was scaled linearly with the following

relationship, where again n is the total number of offspring produced

$$n = i * 30 \tag{5}$$

with values of i ranging from 2 to 10.

Ratio (i)	Max fitness value	Min fitness value	Samples classified as an attack	Samples classified as benign
2	1.715	0.567	12	18
3	1.470	0.641	12	18
4	50.0	0.908	20	10
5	50.0	0.938	21	9
6	8.377	0.987	18	12
7	6.538	1.037	18	12
8	2.174	1.0	21	9
9	5.009	1.218	20	10

Table 7: Ratio Between Offspring Produced and Fittest Offspring Testing Results

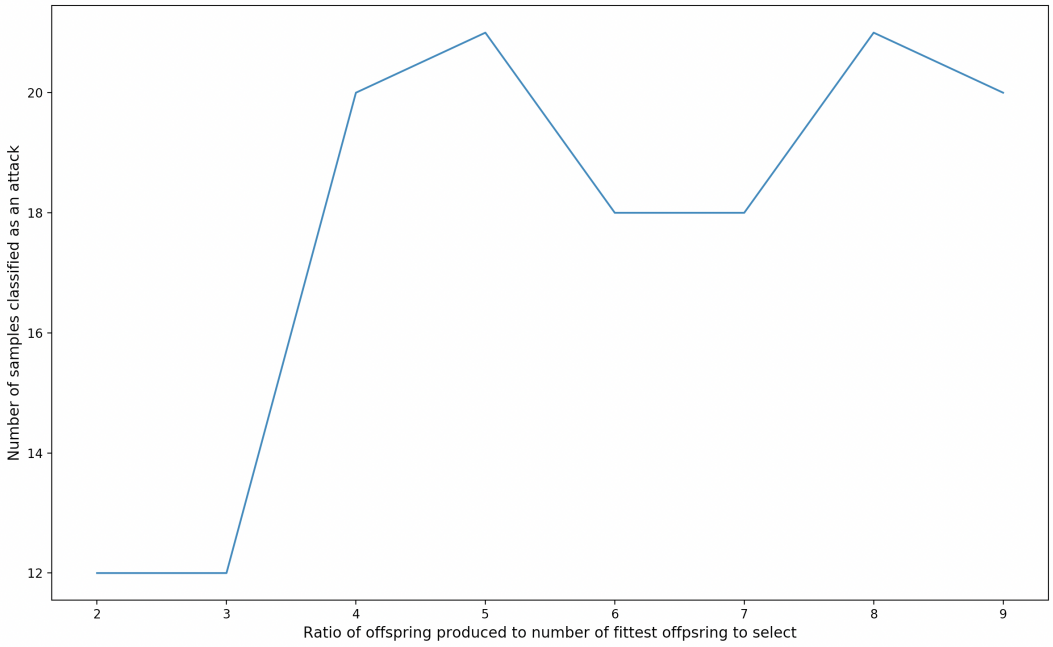

Figure 5: Ratio of number of offspring to fittest offspring vs samples classified as an attack

As can be seen from the above results, the number of samples classified as an attack peaks at around a ratio of 4. After this, the number of attacks remains relatively constant. However the ratio is increased, the running time of the algorithm increases. As such a ratio of 4 was selected for running to keep the run time of the algorithm low while maximising performance.

4.2 Final Parameters and Constants

Based on the data presented in the hyperparameters section, the following are the hyperparameter values used to produce the results presented in the following sections.

- Mutation Percentage: 18%
- Number of Iterations: 20
- Number of offspring per iteration: 120
- Number of fittest offspring to breed next generation: 30

4.3 Algorithm Performance

By running the algorithm five times on each investigated attack type, the following data gives a high-level view of the performance of the genetic algorithm. It is noted that the evasion rate is still a potential evasion rate as the benign samples have not been validated at this point, and only the fittest sample has been analysed in later individual analyses. Some other attack types have also been included to provide some information on the algorithms general performance on attack types that hyperparameters were not tuned for.

Attack	Number of Benign Generated Samples	Number of Attack Generated Samples	Evasion Rate
Teardrop	74	76	49.3%
NMAP	65	85	43.3%
Neptune (DoS)	131	19	87.3%
Loadmodule (U2R)	128	22	85.3%

Table 8: Algorithm Evasion Rate

While the lower evasion rate for the two investigated attacks may first appear that the algorithm performs worse for these attacks, upon analyse of the produced results for the non-investigated attacks, the higher evasion rate

can be attributed to overfitting of the produced samples. This comes down to the fact that while the samples produced are benign, the algorithm has produced samples that differ much more significantly to the actual seed attack, whereas in the teardrop and nmap attacks, while less benign samples are produced, those that are produced are stronger candidates for evasion attacks, as presented below. This can be reduced by tuning the hyperparameters for the specific attack sample being produced.

4.4 Attack Samples

In order to accurately present the results for both the teardrop and nmap attacks, only a single attack generation pipeline run is being considered. This allows the analysis to be more detailed as only a single generated sample with a single seed sample and IDS is being analysed, rather than many at the same time. To select the single pipeline run data that was to be used as part of the analysis, the algorithm was run many times and an average sample was chosen. This average sample is not a statistical average but is based purely on observation of the types of samples seen over the testing of the algorithm, and by picking a suitable average sample.

Another approach that could have been taken would have been to run the algorithm N times and select the best result of the N algorithm runs - with best being defined as picking the run that produced the overall fittest sample, however, this does not accurately reflect the algorithms usual output and as such this method was avoided for this analysis.

4.4.1 Teardrop Attack

For this run of the algorithm, the following is the selected seed attack and the associated fittest sample attack produced at the conclusion of the algorithm.

The seed attack sample is a teardrop attack sample randomly selected from the NSL-KDD dataset and is used as part of the fitness function to evaluate the deviation of produced samples. The attack sample is the fittest sample produced after 20 iterations of the algorithm. This sample was classified as benign by the decision tree classifier, but since it is very similar to the attack sample is considered a candidate evasion attack against the IDS.

Feature Variable	Seed sample value	Generated sample value
duration	0	0
protocol type	UDP	UDP
service	Private	Private
flag	SF	SF
Source Bytes	28	155268078
Destination Bytes	0	0
land	0	0
wrong fragment	3	2
urgent	0	0
hot	0	0
num failed logins	0	0
logged in	0	0
num compromised	0	2594
root shell	0	0
su attempted	0	0
num root	0	0
num file creations	0	14
num shells	0	0
num access files	0	0
num outbound cmds	0	0
is hot login	0	0
is guest login	0	0
count	10	10
srv count	10	2
serror rate	0	0
srv serror rate	0	0
rerror rate	0	0
srv rerror rate	0	0
same srv rate	1	1
diff srv rat	0	0
srv diff host rate	0	0
dst host count	35	35
dst host srv count	10	10
dst host same srv rate	0.29	0.29
dst host diff srv rate	0.11	0
dst host same src port rate	0.29	0.29
dst host srv diff host rate	0	0
dst host serror rate	0	0
dst host srv serror rate	0	0
dst host rerror rate	0	0
dst host srv rerror rate	0	0

Table 9: Teardrop seed and generated attack samples

The following decision tree was also generated as part of the algorithm.

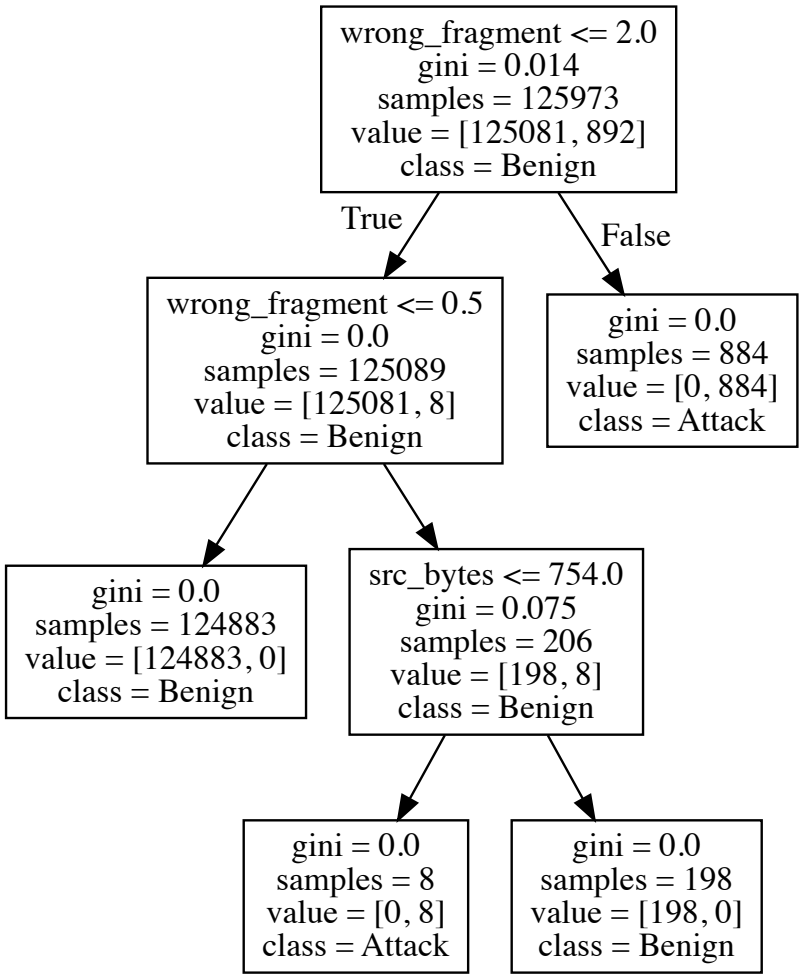

Figure 6: Decision Tree for teardrop attack classification

By comparing the produced sample to the decision tree classifier, it can be seen that the algorithm successfully alters the values that the decision tree

uses as part of its boundary decisions in order to produce a sample similar to the seed attack, but instead of being classified as an attack, it is classified as benign. This process can be seen from the following differences between the seed sample and the generated sample.

- The wrong fragments flag in the produced sample has a value of 2, which is equal to the value at that decision boundary, and as such the left node is selected.
- The wrong fragments value is still greater than the 0.5 value at this decision boundary and as such the right node is selected.
- At the final node, the number of source bytes used in the generated sample is analysed. The cutoff value at this node is 754, and as such the right node is again selected resulting in a benign classification.

4.4.2 NMAP Attack

The following is the produced NMAP attack sample alongside its seed sample, as well as the decision tree classifier produced as part of this run of the algorithm.

Feature Variable	Seed sample value	Generated sample value
duration	0	0
protocol type	TCP	TCP
service	Private	Private
flag	SH	SH
Source Bytes	0	0
Destination Bytes	0	0
land	0	0
wrong fragment	0	0
urgent	0	0
hot	0	0
num failed logins	0	0
logged in	0	0
num compromised	0	0
root shell	0	0
su attempted	0	0
num root	0	0
num file creations	0	0
num shells	0	0
num access files	0	0
num outbound cmds	0	0
is hot login	0	0
is guest login	0	0
count	1	1
srv count	1	1
serror rate	1.0	1.0
srv serror rate	1.0	1.0
rerror rate	0	0
srv rerror rate	0	0
same srv rate	1	1
diff srv rat	0	0
srv diff host rate	0	0
dst host count	16	16
dst host srv count	1	1
dst host same srv rate	0.06	1.0
dst host diff srv rate	1.0	1.0
dst host same src port rate	1.0	1.0
dst host srv diff host rate	0	0
dst host serror rate	1	1
dst host srv serror rate	1	1
dst host rerror rate	0	0
dst host srv rerror rate	0	0

Table 10: NMAP seed and generated attack samples

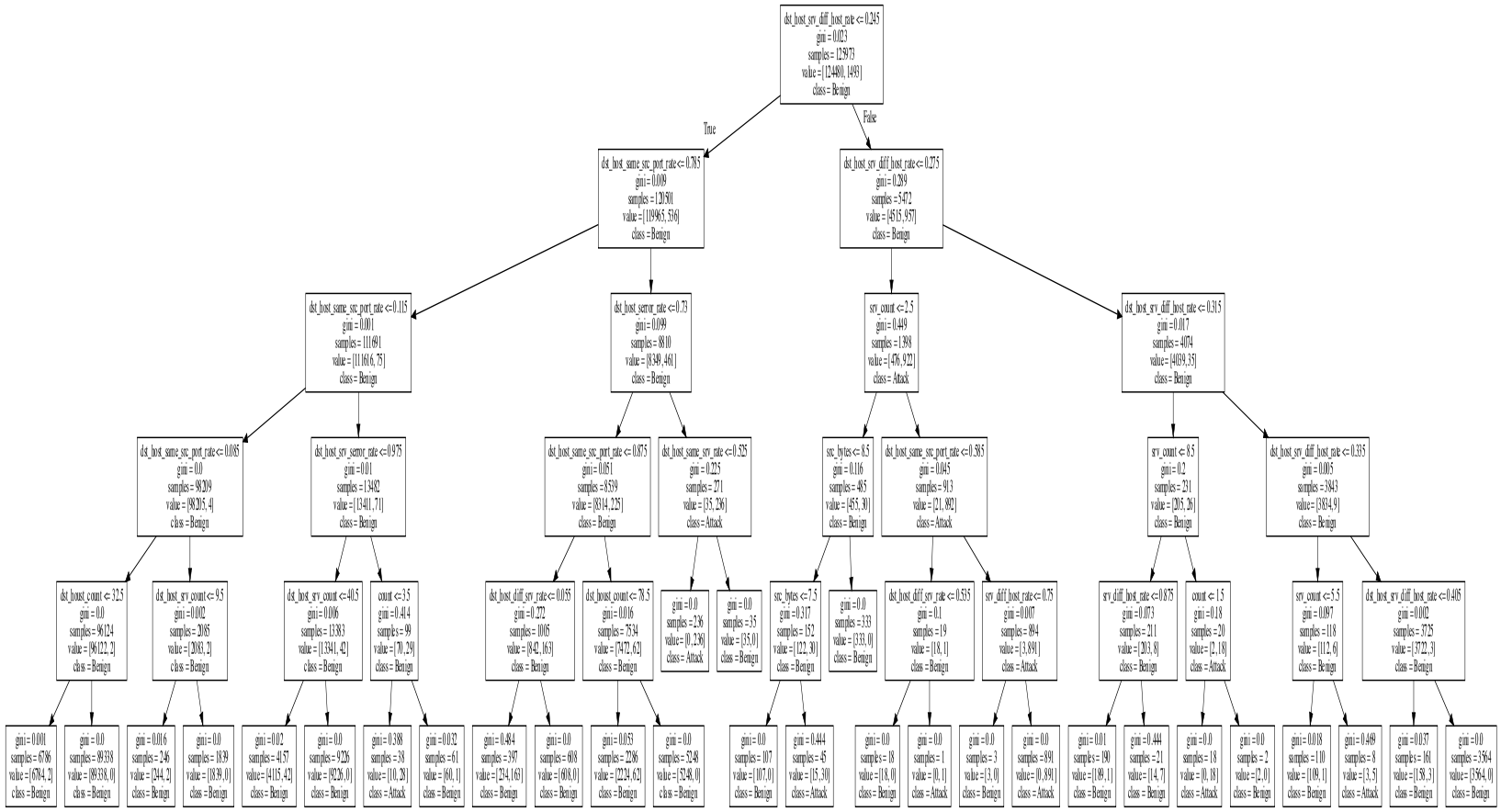

Figure 7: Decision Tree for nmap attack classification

The same process can be applied to nmap attack that was previously used on the teardrop attack to analyse why this produced sample is classified as benign. The following are the decision steps taken to classify the produced sample

- dst host srv diff host rate error for the produced sample is equal to zero, so move left at the first decision boundary as this is less than 0.245
- dst host same src port rate is equal to 1 which is greater than the 0.785 value for the current decision node, so move right.
- dst host serror rate is equal to 1 which is greater than 0.73, so again move right.
- Finally, dst host same srv rate is equal to 1 which is again greater than the decision value of 0.525, leading to a classification of benign.

4.5 Discussion and Limitations

4.5.1 Teardrop

One of the advantages of having a highly interpretable IDS is the fact that analysis along the decision boundaries of the IDS can be performed by eye, as presented in the attack sample section above. This interpretability also allows a comparison of the generated sample to the ideal sample for a single path taken along the IDS. The ideal sample is one that has values exactly corresponding to the decision boundary of the IDS. This is considered an ideal sample as it will give a value closest to where the IDS considers the cutoff for an attack, which means that the sample may still be an attack itself as it will be very similar to a sample classified as an attack, but simply deviates enough to avoid attack classification.

For this produced sample, it can be seen that it correctly alters the wrong fragment feature variable to be in line with the ideal sample value of 2. In the context of a teardrop attack, this value is quite sensible as the aim of the teardrop attack is to send lots of wrong fragments to a host system. Any less than 2 wrong fragments and by definition the attack is no longer a teardrop attack as only a single wrong fragment is sent. The other value used to classify this generated simple is the source bytes feature variable. The source bytes feature variable is not in line with the ideal sample value of 745 and deviates substantially from this value with a value of 155268078 bytes. In the context of a teardrop attack, this could make logical sense however as sending lots of bytes leads to many packets being sent, which if enough wrong fragments are sent would overwhelm a system. Validation of these generated samples and their values in the context of specific attacks is however out of scope for this research, however, some thoughts are presented on this in the further research section of the conclusion.

4.5.2 NMAP

As seen from the results, the produced NMAP decision tree IDS is more complex than the teardrop classifier, and as such analysis of the produced results requires a bit more of an analytical approach.

Again the comparison can be drawn between the IDS and some form of ideal sample for this decision tree, however for the NMAP decision tree there are no longer just a few ideal paths, but many. To simplify this, we will analyse the produced sample against the optimal values along the path it was classified along. All the values modified along the decision path taken by the sample in this classifier are percentage values with a range of 0 to 1. As such we can look at its deviation percentage from each optimal value along that path to get an idea about its performance. For the first feature

variable altered, the dst host srv diff host rate error, the optimal value for benign classification was 0.245, with our sample having a value of 0, giving a difference of 0.245. Applying this same logic to all other variables, we get a total deviation sum of 1.25 out of a max possible deviation sum of 2.795, giving approx a difference of 44% from the optimal value. This is only a deviation percentage for the feature variables that were used to classify the sample, and by looking at the produced sample across the rest of the feature variables, it can be seen it is quite similar to the seed sample.

In context of the attack itself, it can be seen that the produced sample is still viable as an attack. The differencing feature variable in the classification of the sample is the fact that the dst host same srv rate value of the produced sample is 1, compared to the seed attacks value of 0.06. This leads to the sample being classified as benign. Since the dst host same srv rate feature variable identifies the percent of connections that were too different services, this means that all the connections to the dest server, identified by the feature variable dst host count, were to different services. This is logically sound in a NMAP probe where an attacker is attempting to identify what services are running on a given server, and as such the sample seems viable.

4.5.3 Limitations

While the produced results are promising, there are a few limitations to the attack generation pipeline which may limit its usability in real-world scenarios or other research projects.

One limiting feature of this attack generation pipeline is the fact that only basic feature variable dependence is enforced during the generation of attack samples. Some feature variables within the NSL=KDD dataset are highly dependent, such as the number of root actions requiring the logged in as root

flag to be true. While basic variable dependence like this has been enforced, there may be some more complex feature variable relationships within the dataset that are not yet accounted for. This means that running the algorithm may still produce samples that are not technically feasible.A more complex restraint on how mutation is applied across feature variables may further increase the accuracy of produced samples.

Another potentially limiting factor is the assumptions that go into producing the fitness function. It was assumed that by producing samples that are similar to a known attack but have some deviation in certain feature variables, an evasion attack can be produced. This assumption however may not hold up in sample validation as there is the chance that for given attacks, modifying the sample may render the sample either non-valid or no longer an attack at all. This assumption may also limit the "creativity" of the attack pipeline. Attacks are only able to be generated mimicking closely a known attack, however, a malicious actor may be able to craft an attack with a very different pattern. Given an attack validation component, the restraints on the fitness function can be modified to no longer require this deviation constraint, which may in turn produce better results.

Finally, the measure of deviation and use of a single seed sample can be considered a limitation of the genetic algorithm pipeline. Since a single seed sample is used as part of the fitness function, all samples are limited to attempting to minimise the deviation from this sample. However, this may not be the most optimal way of producing samples as attacks in themselves can differ substantially across a single attack type. By using some aggregated method to seed the algorithm, improved attack samples may be able to be produced.

4.5.4 Recommendations for IDS Design

While the presented results only show attacks being generated against a decision tree based IDS, some conclusions and recommendations can be drawn from what has been produced.

One recommendation that can be drawn from the presented research stems from the fact that for the pipeline to operate at all a feedback loop is required between the targeted IDS and the genetic algorithm itself. This means that the output of the IDS should never be exposed in any way, besides to internal security teams. Should a malicious actor have access to the IDS output, they will be able to attempt to generate attacks by using this output as a feedback loop using the presented methods. This also then leads to the fact that while interpretability is important for IDS designers and security teams, by obfuscating the internal workings of the IDS, it will be less susceptible to attack. A malicious user wishing to attack an IDS who has some knowledge about the system can attempt to closely model it and use the produced model as part of the attack generation pipeline. The model may not need to accurately represent the real world IDS, but simply be close enough to be able to generate attacks. While this could be considered "security by obscurity" which is generally bad practice, by protecting this information the chance of an attack being generated against the system is lessened.

Another recommendation for the design of IDSes that is obvious from the output of the attack generation pipeline is for IDSes to flag inputs to the IDS that are extremely similar in content and occurring close together. By using a similar method to the attack generation pipeline for measuring deviation, inputs can be evaluated against recently received inputs. Since the attack generation pipeline generates attacks that are very similar to a known attack, the output potential attacks are all quite similar in structure. A malicious

actor may have to attempt to use a few of the produced attacks before a successful one is found. By flagging inputs that look similar in structure, this scenario can be detected and prevented.

5 Conclusions and Future Work

As presented, interpretable evasion attacks have been generated against a decision tree based IDS using genetic algorithms, as such meeting the initial aims and goals this research set out to achieve. A generalised and open-sourced attack pipeline has been proposed and constructed that allows anyone to generate evasion samples using the same genetic algorithm approach, given the attack is one present in the NSL-KDD dataset.

The samples produced were analysed and found to be similar to a given seed attack sample across both the teardrop and NMAP attack types. This indicates that the genetic algorithm fitness function was designed successfully to meet its goal of producing samples similar to a given seed attack while being classified as benign. Due to the fact that these samples were similar to a given attack but were classified as benign, they serve as good candidates for potential evasion attacks. Further validation needs to be done on the produced samples to ensure that they are still attacks and can be executed against some host system, and as such were misclassified by the IDS itself.

In terms of the applications of the produced results, this research shows that there is the potential for behaviour based IDSes such as the decison tree based IDS presented, to be attacked by genetic algorithms. While the presented IDS is only a single decision tree, the attack pipeline can be generalised to plug in any IDS possible by simply conforming to the interface defined in the pipeline. As such this presented attack generation pipeline could immediately be used to attempt to generate attacks against industry level IDSes, given a feedback loop from the IDS is present. Since genetic algorithms learn over time and utilise this feedback loop from the IDS itself, it is not far fetched to predict that such attacks could be generated against

more complex IDSes.

This research only sets a base level for generating evasion attacks against an IDS and as such could be built on top of to give more insight into the performance of IDSes in industry. The first major area of future work would be working on the validation of produced evasion samples. The idea that producing samples that are as similar as possible to a sample attack may work, but there needs to be some actual validation to prove this. For example, take the produce teardrop attack samples which dropped the number of wrong fragments sent from 3 to 2. While this may still function as a teardrop attack, sending 2 wrong fragments may fundamentally not be enough to overwhelm a host system, and as such the produced sample may no longer be an attack at all, meaning that the IDS correctly classified the sample. To validate samples, a sandbox system could be used to run the samples through and evaluate the results on various host systems. This could then in turn be plugged into the attack generation pipeline as a separate component, further increasing the power of the attacks it could generate as this feedback could be used as part of an improved fitness function.

Another potential area of further research would be testing the attack generation pipeline on other IDS designs and analysing its performance. While a decision tree based IDS provides great interpretability, most industry IDSes are hybrids [1], and as such testing attack generation on other IDSes may provide more insight into their weaknesses and how they could be improved.

Overall, the results of the presented research are promising. Further work could provide even more valuable insight that could be utilised to vastly improve current intrusion detection systems.

6 Appendix

6.1 Appendix A - Codebase

All code used to generate results as part of this research can be found at:

https://github.com/raymogg/GeneticAlgorithmIDSAttacks

The interface section and repository README.md file provides more detail on using the genetic algorithm from your own python scripts.

References

[1] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.

[2] "10 cyber security facts and statistics for 2018," Available at https://us.norton.com/internetsecurity-emerging-threats-10-facts-about-todays-cybersecurity-landscape-that-you-should-know.html (2018/01/01).

[3] I. Sharafaldin., A. H. Lashkari., and A. A. Ghorbani., "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, INSTICC. SciTePress, 2018, pp. 108–116.

[4] M. Tavallaee., E. Bagheri., W. Lu., and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," *IEEE symposium on computational intelligence in security and defence applications*, 2009.

[5] "What are the most common cyberattacks?" Available at https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html (2019/07/07).

[6] P. Technologies, "Cybersecurity threatscape 2018: trends and forecasts," Moscow, Russia, Tech. Rep., 2019.

[7] M. T. Ribeiro, S. Singh, and C. Guestrin, "Model-agnostic interpretability of machine learning," 2016.

[8] A. Patel., M. Taghavi., K. Bakhtiyari., and J. C. Júnior, "Taxonomy and proposed architecture of intrusion detection and prevention systems for

cloud computing," in *Cyberspace Safety and Security. Lecture Notes in Computer Science, vol 7672*. Springer, 2012, pp. 441–458.

[9] M. W. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained networks," in *Proceedings of the 8th International Conference on Neural Information Processing Systems*, ser. NIPS'95. Cambridge, MA, USA: MIT Press, 1995, p. 24–30.

[10] E. trumbelj and I. Kononenko, "An efficient explanation of individual classifications using game theory," *J. Mach. Learn. Res.*, vol. 11, pp. 1–18, 2010.

[11] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues," *Information Sciences*, vol. 239, p. 201, 2013.

[12] D. L. Hudson and M. E. Cohen, "Genetic algorithms," in *Neural Networks and Artificial Intelligence for Biomedical Engineering*. Hoboken, NJ, USA: John Wiley Sons, Inc., 2012, pp. 215–224.

[13] Z. Lin, Y. Shi, and Z. Xue, "Idsgan: Generative adversarial networks for attack generation against intrusion detection," 2018.

[14] N. R. Rodofile, "Generating attacks and labelling attack datasets for industrial control intrusion detection systems," 2018. [Online]. Available: https://eprints.qut.edu.au/121760/

[15] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. New York, NY, USA: ACM, 2002, pp. 255–264. [Online]. Available: http://doi.acm.org/10.1145/586110.586145